

## **SYSC4001A L3 Group 2, Assignment 1**

### **Part 2: Report**

**Repository Link:** <https://github.com/celinaayangg/SYSC4001-Assignment-1>

#### **1. Change the value of the save/restore context time from 10, to 20, to 30ms. What do you observe?**

For 10 ms to 20 ms: since each interrupt handling (both SYSCALL and END\_IO) does one save and one restore, total overhead doubles. For 10 ms to 30 ms, total overheads triples. For example, in trace.txt, we do 30x SYSCALL and 30x END\_IO for a total of 60 interrupt handlings. 10 ms to 20 ms adds 20 ms per interrupt for a total of 1200 ms for the whole run. Further increasing to 30 ms adds 2400 ms to the whole run.

However, the event order is unchanged. There is more time in kernel overhead, the proportion of time spent doing useful CPU work shrinks and overall turnaround increases. Since we assumed that I/O devices are always and constantly available, the I/O started immediately upon request, thus adding no extra latency. So completion timing is dictated purely by the trace.

#### **2. Vary the ISR activity time from between 40 and 200, what happens when the ISR execution takes too long?**

As the ISR activity time increases, it slows down the entire system. Going from 40 ms to 200 ms raises the latency by 160 ms per interrupt. For 60 interrupts, that's an extra 9600 ms overall. This delay causes the system to remain in kernel mode for a longer period, causing the following steps to wait until the ISR execution is complete. As a result, the activity prevents the operating system from converting back to user mode until the execution is complete, increasing the overall execution time and reducing the efficiency of the CPU. In execution.txt, all events would shift to be later, every SYSCALL and END\_IO that follows would be affected by the accumulated extra ISR time. Also, if we assume our system is a single CPU system, it can only run either user code or the ISR. Interrupts are masked during the ISR, equal or lower priority interrupts would be blocked.

#### **3. How does the difference in speed of these steps affect the overall execution time of the process?**

Based on the results of the simulation, it shows that some of the steps take a longer execution time and as a result causes the overall execution time to take longer to complete. The faster steps within the simulation include: switching to and from kernel mode, calculating the ISR start address, getting the ISR address, and executing IRET all result in a 1ms time duration. Some of the more time consuming steps include saving and restoring context and executing the ISR body. The saving and restoring context step of the simulation results in an approximate duration of 10ms whereas executing the ISR body results in a range of approximately 40ms to 200ms. When

these steps execute, there is a noticeable increase in execution time considering the amount of times these steps occur within the simulation. The CPU burst at the beginning of the system call also contributes to the overall execution time and based on the trace.txt, the runtime ranges from 11ms to 100ms. These bursts represent the time that occurs between interrupts including the time between “SYSCALL” and “ENDIO”. The most time consuming step within the simulation is checking for errors. This is because this step represents the I/O delay associated with each device given in the device\_table.txt. This step displays the hardware's response time and given the device\_table.txt, the range is from 68ms to 1000ms.

**4. You can process this data using a Python script, Excel spreadsheet or any other tools for separating the overhead from the actual work of your program (i.e., CPU use for processing and actual I/O needed by the program).**

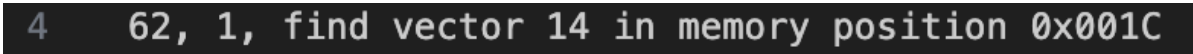
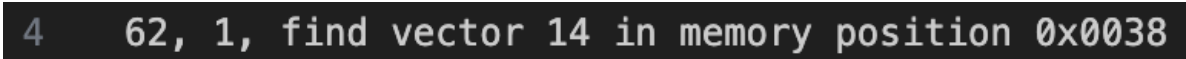
0	62	CPU burst	Actual Work
62	1	switch to kernel mode	Overhead
63	10	context saved	Overhead
73	1	find vector 9 in memory position 0x0012	Overhead
74	1	load address 0X036C into the PC	Overhead
75	40	SYSCALL: run the ISR (device driver)	Actual Work
115	40	transfer data from device to memory	Actual Work
155	156	check for errors	Actual Work
311	1	executing IRET	Overhead
312	1	switch to kernel mode	Overhead
313	10	context saved	Overhead
323	1	find vector 9 in memory position 0x0012	Overhead
324	1	load address 0X036C into the PC	Overhead
325	40	ENDIO: run the ISR (device driver)	Actual Work
365	156	check device status	Actual Work
521	1	executing IRET	Overhead
522	49	CPU burst	Actual Work
571	1	switch to kernel mode	Overhead
572	10	context saved	Overhead
582	1	find vector 15 in memory position 0x001E	Overhead
583	1	load address 0X0584 into the PC	Overhead
584	40	SYSCALL: run the ISR (device driver)	Actual Work
624	40	transfer data from device to memory	Actual Work
664	68	check for errors	Actual Work
732	1	executing IRET	Overhead
733	1	switch to kernel mode	Overhead
734	10	context saved	Overhead
744	1	find vector 15 in memory position 0x001E	Overhead
745	1	load address 0X0584 into the PC	Overhead
746	40	ENDIO: run the ISR (device driver)	Actual Work
786	68	check device status	Actual Work

Implementing the execution.txt to excel and to compare the overhead and actual work of the program.

**5. Ask yourselves other interesting questions and try to answer them through simulations. For instance: what happens if we have addresses of 4 bytes instead of 2? What if we have a faster CPU?**

**i) What happens if we have addresses of 4 bytes instead of 2?**

When running the simulation using 4 bytes instead of 2, there is no observable change that affects the overall execution time of the simulation. Since both steps, calculating the ISR start address and getting the ISR address remains a constant 1ms of response time and therefore, the change in the byte has no impact on the overall execution time. The only visible difference seen is the memory value of the hexadecimal string has changed.

1. 
2. 

Based on image 1 (2 bytes) and image 2 (4 bytes) the memory position address has changed from “0x001C” to “0x0038.”

**ii) What if we have a faster CPU?**

Having a faster CPU would result in a more efficient performance of the program. This is because the steps that are CPU dependent would result in a change in the steps execution time. Steps such as saving and restoring context and executing the ISR body have higher response times and having a faster CPU would reduce the duration, affecting the overall execution time.